AI Client APIs Documentation

This document describes the APIs available for interacting with various AI providers (DeepSeek, Google Gemini, and OpenAI).

Common Interface

All providers implement the following interface for making queries. See pkg/sqirvy/client.go for the full interface definition.

```go
// pkg/sqirvy/client.go
const (
    Anthropic Provider = "anthropic" // Anthropic's Claude models
    DeepSeek  Provider = "deepseek"  // DeepSeek's models
    Gemini    Provider = "gemini"    // Google's Gemini models
    OpenAI    Provider = "openai"    // OpenAI's GPT models
    MetaLlama Provider = "llama"     // Meta's Llama models
)

type Options struct {
    Temperature float32 // Controls randomness (0-100)
    MaxTokens   int64   // Maximum tokens in response
}

type Client interface {
    QueryText(ctx context.Context, system string, prompts []string, model
string, options Options) (string, error)
}

func NewClient(provider Provider) (Client, error)
```

Usage Example

See the code in directory 'examples' for complete examples of using the client APIs.

```go
model := "gemini-pro"
systemPrompt := "you are a helpful chatbot"
userPrompts := []string{"What is the meaning of life?"}

// Create a new client
client, err := NewClient(sqirvy.Gemini)
if err != nil {
    log.Fatal(err)
}

// Configure options
options := Options{
    Temperature: 50,    // 50% temperature
    MaxTokens: 8192,    // Default token limit
}

// Query for text
response, err := client.QueryText(ctx, systemPrompt, userPrompts, model,
options)
if err != nil {
    log.Fatal(err)
}
```

Error Handling

All methods return errors in the following cases:

- Missing API keys
- Empty or invalid prompts
- Invalid temperature values (must be 0-100)
- API request failures
- Invalid responses

Environment Variables

The following environment variables are used:

- `ANTHROPIC_API_KEY` - For Anthropic Claude API access
- `DEEPSEEK_API_KEY` and `DEEPSEEK_BASE_URL` - For DeepSeek API access
- `GEMINI_API_KEY` - For Google Gemini API access
- `LLAMA_API_KEY` and `LLAMA_BASE_URL` - For Meta Llama API access
- `OPENAI_API_KEY` - For OpenAI API access

Provider-Specific Implementations

DeepSeek Client

The DeepSeek client interfaces with DeepSeek's models.

Models

- Tested with: `deepseek-coder`, `deepseek-chat`

Features

- Temperature scaled to 0-2.0 range
- Default max tokens: 8192
- Requires both API key and base URL
- Returns error if prompt is empty

Google Gemini Client

The Gemini client interfaces with Google's Gemini models.

Models

- Tested with: `gemini-pro`, `gemini-pro-vision`

Features

- Temperature scaled to 0-2.0 range
- Uses generative model with text/plain MIME type
- Concatenates multiple response parts
- Returns error if prompt is empty

OpenAI Client

The OpenAI client interfaces with GPT models via the OpenAI API.

Models

- Tested with: `gpt-4`, `gpt-4-turbo`, `gpt-3.5-turbo`

Features

- Temperature scaled to 0-2.0 range
- Default max tokens: 8192
- Returns error if prompt is empty
- Supports custom base URL via environment variable

Utility Functions

The package also provides utility functions in pkg/util for:

File Operations

```go
// Read from stdin if available
func InputIsFromPipe() (bool, error)
func ReadStdin(maxTotalBytes int64) (data string, size int64, err error)

// Read from files
func ReadFile(fname string, maxTotalBytes int64) ([]byte, int64, error)
func ReadFiles(filenames []string, maxTotalBytes int64) (string, int64,
error)
```

Web Scraping

```go
// Scrape content from URLs
func ScrapeURL(link string) (string, error)
func ScrapeAll(urls []string) (string, error)
```

These utilities handle:

- File path validation and cleaning
- Size limit enforcement
- Error handling for missing/invalid files
- URL validation and scraping
- Content formatting with Markdown code blocks